Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Stealth attacks: An extended insight into the obfuscation effects on Android malware

Davide Maiorca[*], *Davide Ariu*, *Igino Corona*, *Marco Aresu*, *Giorgio Giacinto*

Department of Electrical and Electronic Engineering, University of Cagliari, Piazza d'Armi, 09123, Cagliari, Italy

ARTICLE INFO

Article history:
Received 5 August 2014
Received in revised form
22 December 2014
Accepted 24 February 2015
Available online 14 March 2015

Keywords:
Android
Malware
Obfuscation
Evasion
DexGuard
Dalvik
Entry points
Signatures
Strings
Bytecode

ABSTRACT

In order to effectively evade anti-malware solutions, Android malware authors are progressively resorting to automatic obfuscation strategies. Recent works have shown, on small-scale experiments, the possibility of evading anti-malware engines by applying simple obfuscation transformations on previously detected malware samples. In this paper, we provide a large-scale experiment in which the detection performances of a high number of anti-malware solutions are tested against two different sets of malware samples that have been obfuscated according to different strategies. Moreover, we show that anti-malware engines search for possible malicious content inside assets and entry-point classes. We also provide a temporal analysis of the detection performances of anti-malware engines to verify if their resilience has improved since 2013. Finally, we show how, by manipulating the area of the Android executable that contains the strings used by the application, it is possible to deceive anti-malware engines so that they will identify legitimate samples as malware. On one hand, the attained results show that anti-malware systems have improved their resilience against trivial obfuscation techniques. On the other hand, more complex changes to the application executable have proved to be still effective against detection. Thus, we claim that a deeper static (or dynamic) analysis of the application is needed to improve the robustness of such systems.

1. Introduction

Not surprisingly, malware writers are paying more and more attention to mobile devices. In fact, the number of mobile devices sold worldwide has already surpassed that of traditional personal computers. According to a recent report by F-Secure, more than 99% of the new mobile malware families discovered in 2014 targets the Android platform, which accounts for more than 750 millions of active devices (F-Secure, March 2014).

There are several reasons for which Android is a particular interesting target for deploying malware:

1. Its open source nature allows an attacker to carefully study the operating system implementation, thus increasing the probability of finding vulnerabilities.
2. There are multiple alternative markets besides the official one (Google Play), in which it is possible to find applications that are not released through the support of Google (for example, for copyright reasons) or to find popular premium applications at a reduced price. Popular examples are the

* *Corresponding author*.
E-mail addresses: davide.maiorca@diee.unica.it (D. Maiorca), davide.ariu@diee.unica.it (D. Ariu), igino.corona@diee.unica.it (I. Corona), arsu.ma@gmail.com (M. Aresu), giacinto@diee.unica.it (G. Giacinto).
http://dx.doi.org/10.1016/j.cose.2015.02.007
0167-4048/

Amazon or Samsung app stores (Amazon App Stores; Samsung App Stores). However, many of these markets provide insufficient control on the security of the applications, thus becoming the first source of mobile malware (Comparatives).

3. Even if Google Play features an advanced dynamic analysis system for detecting malware and spyware, called Google Bouncer, several malware have managed to bypass it even very recently (see, for instance, Labs; BGR; SecurityWatch).
4. The problem gets even more serious because of the bad management of digital signatures in the Google Play store. In order to upload an app to the store and install it on the phone, an application must be signed with a certificate. This certificate is also used as a reference for other applications to *share* their data. Sadly, many legitimate applications are signed with extremely insecure private keys (i.e., private keys that can be easily obtained by an attacker). It is therefore not so difficult for an attacker to digitally sign a malicious application so that its certificate would look trusted. In this way, it is even possible to *replace* a legitimate app with its malicious variant through an update process (Palo Alto Networks). Recently, a dangerous vulnerability that exploited the lack of control in the certificate issuer by the Android cryptographic system has been discovered, and it has been there until the most recent versions of Android, including KitKat (4.4) (Bluebox, January 2010).

To improve the security of the users, a number of different anti-malware solutions have been developed. These solutions perform an in-depth scan of the application, including its bytecode, external resources such as images, audio and so forth. Consequently, Android malware are getting more sophisticated, not only because of the different types of attacks that they can implement (e.g., root exploits, embedded executables, invisible layouts, encrypted C&C communications, etc. (Jiang, 2011; Labs, 2011), but also for the possibility of deceiving reverse engineering attempts or anti-malware analysis through *obfuscation* (Unuchek, June 2013; Ionescu, June 2012). In this paper, we refer to the term obfuscation as actions that perform changes on the application while preserving its semantics. For instance, they can modify its bytecode, strings, or resource files. The aim of obfuscation is making applications more difficult to be analyzed by humans or automatic tools.

Obfuscation can be used to protect applications from being plagiarized or cloned. However, some obfuscation strategies might also be used to easily create new versions of the same malware that are more difficult to analyze. The attacker is motivated to adopt them, as automatic analysis tools often rely on *static signatures* that can be easily evaded by changing few elements of the applications (for example, replacing the name of the methods). It is possible to find different examples of obfuscation in the wild, such as those reported in (Yu, 2013; Apvrille & Nigam; Ballano). A number of automatic tools, available either as commercial products or for free, can be used in order to ease malware obfuscation (Lafortune; Saikoa).

Previous works explored the world of obfuscation for the Android platform, by pointing out how specific obfuscation techniques can be effective to evade popular anti-malware solutions (Zheng et al., 2012; Rastogi et al., 2013). In particular, the work by Rastogi et al. (2014) clearly showed how anti-malware systems are weak against easy-to-implement transformation techniques. In particular, it showed that it is possible to evade the vast majority of the most popular anti-malware software for Android by applying a combination of obfuscation techniques. It also showed that many anti-malware software tend to rely on signatures that are weak and easy to bypass. That work was carried out by testing obfuscation techniques against six malware samples, where anti-malware signatures were updated at the beginning of 2013.

1.1. *Contributions*

In this paper, we provide a deeper insight into the effects of the obfuscation of Android malware. First, our interest is to assess the current status of the anti-malware detection capabilities, as almost one year has passed since the analysis made by Rastogi et al. We do so by deploying a *large-scale* experiment on more than 50 malware families and two malware datasets, namely, Malgenome and Contagio, for a total of more than 1200 samples (Zhou and Jiang, 2012; Parkour). We obfuscate the malware samples in these datasets by means of different strategies, which differ from each other in terms of complexity (from simple ones such as class and methods renaming, to complex ones such as reflection and class encryption), and areas of the Android executable that are targeted (e.g., *strings*, *bytecode*, or both). We experiment new obfuscation strategies, and their combinations that have never been tested in previous works (such as the combination of obfuscation by Reflection and Class Encryption). Our tests have been carried out by running 13 among the most popular anti-malware solutions available on the Android market Google Play. This experimental set-up provides, to the best of our knowledge, the biggest assessment of anti-malware performances against obfuscated samples in comparison with previous works.

The second contribution of this work concerns the assessment of the parts of the application that might be decisive in the anti-malware detection process. To this end, we focused on the incidence of external resources, such as *assets*. We found that anti-malware engines resort to analyzing and flagging external resources as malicious as an aid for the detection, thus confirming the findings in previous works. However, we also point out the extent to which the analysis of external resources plays a key role in the final outcome produced by anti-malware tools. To this end, we show that by manipulating the assets of a malware sample it is possible to evade the detection. We will also explain, though, that while the ad-hoc manual implementation is quite easy to deploy, its automatic version is quite difficult to develop.

We also examine the role of *entry-point* classes in the detection outcome of anti-malware systems detection. In particular, we show that many anti-malware engines rely on the analysis of such classes in order to perform malware detection. While obfuscating such classes is not trivial, as it might lead to completely break the application, we believe that such a choice is reasonable, to the extent to which other obfuscation strategies fail at evading detection.